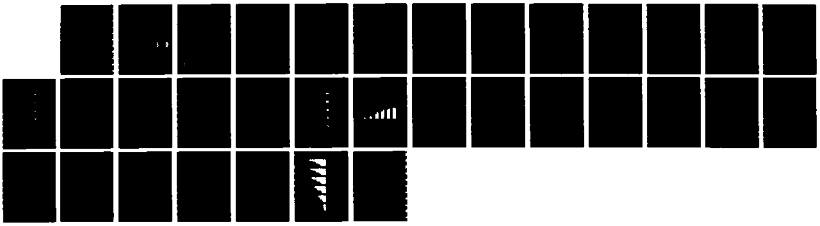
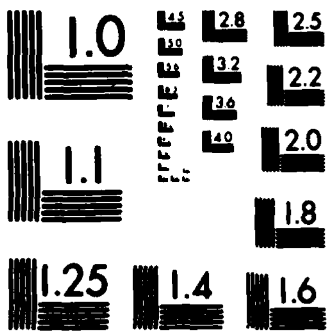
AD-A171 898 BLOCK-ORIENTED LOCAL-MEMORY LINEAR EQUATION SOLUTION ON THE CRAY-2 PART 1..(U) MICHIGAN UNIV ANN ARBOR SUPERCOMPUTER ALGORITHM RESEARCH LAB.. D A CALAHAN 1/1

UNCLASSIFIED 15 DEC 85 SARL-9 AFOSR-TR-86-0681 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

2

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 86-0681 / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) BLOCK_ORIENTED, LOCAL MEMORY LINEAR EQUATION SOLUTION ON THE CRAY-2 PART I: UNIPROCESSOR ALGORITHMS	5. TYPE OF REPORT & PERIOD COVERED Interim
	6. PERFORMING ORG. REPORT NUMBER SARL #9
7. AUTHOR(s) D. A. Calahan	8. CONTRACT OR GRANT NUMBER(s) AFOSR 84-0096
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Michigan Dept. of Elec. Engring. & Comp. Science Ann Arbor, MI, 48109	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ~~9749-03~~ 61102F 2304 A5
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research Bolling AFB, DC, 20332	12. REPORT DATE December 15, 1985
	13. NUMBER OF PAGES 25
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

DTIC SELECTE SEP 1 5 1986 D

S E

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Presented at 2nd SIAM Conf. on Scientific & Parallel Computing, Norfolk, Nov. 18, 1985

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Supercomputers
Parallel.processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Experience with the CRAY-2 on the effects of common memory speed and loading on performance indicate that local-memory-based algorithms have potentially a large advantage. The performance of a number of common- and local-memory algorithms are compared for the LU factorization of a dense system of equations on the CRAY-2. Results of both Fortran and assembly language implementations are given.

REPORT SARL #9

BLOCK-ORIENTED, LOCAL-MEMORY LINEAR EQUATION SOLUTION ON THE CRAY-2.

PART I: UNIPROCESSOR ALGORITHMS

D. A. CALAHAN

DECEMBER 15, 1985

Accession For	
NTIS GRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

SPONSORED BY

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
UNDER GRANT AF-AFOSR-84-0096

SUPERCOMPUTER ALGORITHM RESEARCH LABORATORY
DEPARTMENT OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE

BLOCK-ORIENTED, LOCAL-MEMORY LINEAR EQUATION SOLUTION ON THE CRAY-2.

PART I: UNIPROCESSOR ALGORITHMS

D. A. CALAHAN

DECEMBER 15, 1985

SPONSORED BY

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
UNDER GRANT AF-AFOSR-84-0096

SUPERCOMPUTER ALGORITHM RESEARCH LABORATORY
DEPARTMENT OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE
UNIVERSITY OF MICHIGAN

ABSTRACT

Experience with the CRAY-2 on the effects of common memory speed and loading on performance indicate that local-memory-based algorithms have potentially a large advantage. The performance of a number of common- and local-memory algorithms are compared for the LU factorization of a dense system of equations on the CRAY-2. Results of both Fortran and assembly language implementations are given.

ACKNOWLEDGEMENT

High-performance CAL kernels used in the equation solvers were developed by Geoffrey Carpenter, U. M. Fayyad, and Jimmy Hsiao with a CRAY-2 instruction-level timing simulator developed by K. B. Elliott, as part of a CRAY-2 scientific library sponsored in part by NASA Ames Research Center [11].

CRAY-2 time was provided by MFECC, Lawrence Livermore National Laboratory.

I. INTRODUCTION

A. CRAY-2 Architecture and Algorithm Implications

The CRAY-2 architecture of Figure 1 has several features relevant to this algorithm study.

(a) Common memory features. The massive common memory (CM) trades size for access time, so that a considerable delay is encountered in reading from CM. Also, only one data path connects common memory to each processor's functional units.

(b) Local memory. The above speed disadvantages are compensated by a local memory (LM), which serves as backup vector and scalar storage for the functional unit's register storage.

(c) Chaining. The CRAY-2 does not have hardware chaining; this must be achieved by software and/or algorithm means.

The implications of distributed memory (including hierarchies such as CM and LM) on linear algebra algorithm organization has been studied since the existence of paged memory systems [1[2][3][4]. In general, computations must be arranged so that the number of floating-point operations on data at the low memory levels is sufficient to warrant data transfers to these levels. This implies, for example, that a matrix-vector multiply - which involves only two operations for each matrix data element - will perform less efficiently than a matrix-matrix multiply.

B. Review of Vector Linear Algebra Algorithms

The asymptotic execution rate (MFLOPS) of a factorization algorithm is equal that of the kernel that performs the add-multiplies associated with reducing rows and columns. Three substantially different such algorithms deserve consideration.

(a) Gauss vector-scalar multiply. This requires that, in reducing the rth row , successive operations on preceding rows must be performed serially since a partial result from each row-operation is used as an operand in the next one (the reader is assumed familiar with this procedure). In rows with lengths longer than the vector functional unit length, this dependency can usually be avoided by assembly coding; it is then termed the GAXPY kernel. It has the advantage of yielding the largest average vector length of any of the following kernels and so is a serious consideration when the matrix size is not significantly larger than the maximum allowable vector length, and assembly coding is allowed. This procedure does not lend itself to partitioning when large matrices are involved, but is a potentially useful subalgorithm in such cases.

(b) Matrix-vector multiply. Early experience with the CRAY-1 indicated that block-oriented algorithm organization had at least a pedagogical advantage for large problems[5][6]. Unfortunately, the necessary {vector - (matrix*vector)} kernel was not made a part of the CRAY scientific library and consequently the kernel was not syntactically distinguised from the rest of CAL-coded factorization algorithms. The organizational concept of basing factorization on matrix-vector multiply subroutines was developed in [7][8], where it was illustrated how unrolling Fortran loops could be used to achieve a high performance completely from a high-level language. This emphasized portability and maintainability. More recently, these kernels - known as second-level BLAS - have been proposed as the basis for other common linear algebra algorithms[9].

(c) Matrix-matrix multiply. Although it has always been clear that factorization can be accomplished by a matrix-matrix multiply kernel, the CRAY-1 memory hierarchy was not sufficiently distinctive to achieve a significant advantage over the kernels of (a) and (b) above [3]. The additional memory paths of the X-MP made this even less attractive, and partially reduced the advantage of the matrix-vector multiply above. The disadvantages of basing factorization on matrix multiplies are the necessity for other matrix-level kernels to perform reciprocations and substitutions and, most important, the difficulty of partial pivoting.

The memory distribution must be quite distinctive to warrant the prog.amming effort of (c). This report documents this case for the CRAY-2, while laying the groundwork for a multiprocessor implementation.

II. NON-PIVOTING ALGORITHMS AND PERFORMANCE

A. M*V-based Algorithms

Given a set of equations*

$$\mathbf{A} X = B$$

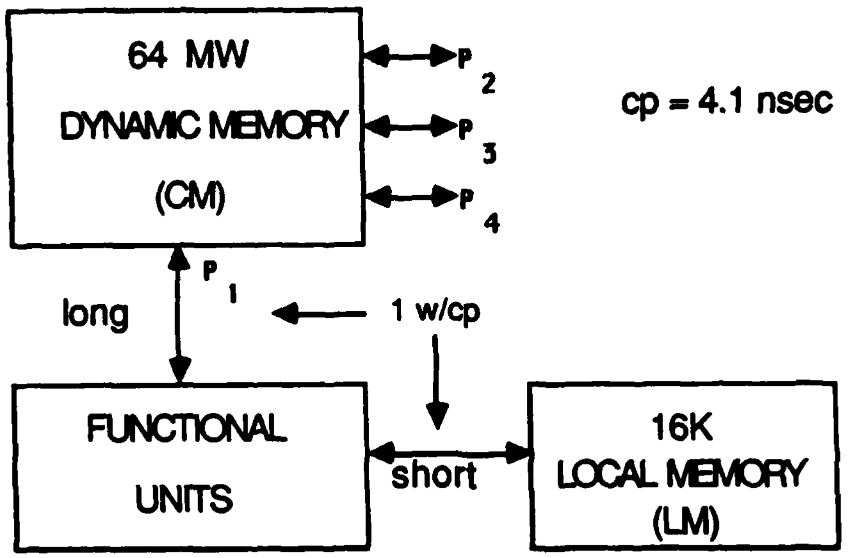
where **A** is an nxn matrix and X and B are vectors, the factorized solution proceeds by forming lower and upper triangular factors **L** and **U**, viz

$$\mathbf{A} = \mathbf{L}\,\mathbf{U} \qquad (1)$$

and then solving

$$\mathbf{L} Y = B$$

*Matrices are in bold, vectors are in upper case, and scalars are in lower case type.



488 MFLOPS peak / processor

459 MFLOPS attainable / processor

Figure 1. CRAY-2 architecture

for Y and

$$U X = Y$$

for X. The complexity of the factorization step of Eq. (1) is $O(n^3)$. The other steps have complexity $O(n^2)$, with only one add-multiply for each **L** and **U** element; consequently no algorithmic speedup results from transferring **L** and **U** to local memory, so that these steps, if performed separately from the factorization as above, will not be studied.

In the non-pivoting algorithms based on a matrix-vector multiply (abbr. M*V-based), the columns of **L** and rows of **U** are indicated as in Figure 2a. Here the diagonal element, the row to its right, and the column below it are denoted a_{22}, A_{12}, and A_{21} respectively. The steps to perform the factorization are then

Matrix-vector multiplies:

$$\begin{vmatrix} a_{22} \\ A_{32} \end{vmatrix} \;\texttt{<---}\; \begin{vmatrix} a_{22} \\ A_{32} \end{vmatrix} - \begin{vmatrix} A_{21} \\ A_{31} \end{vmatrix} \; A_{12} \qquad (2)$$

$$A_{23} \;\texttt{<---}\; A_{23} - A_{21} \; A_{13} \qquad (3)$$

Reciprocation:

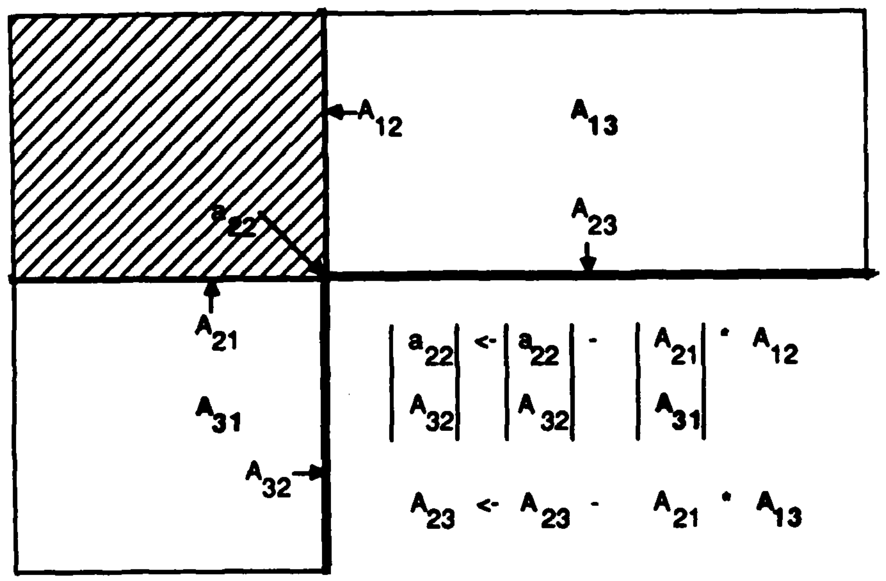
$$a_{22} \;\texttt{<---}\; 1/a_{22} \qquad (4)$$

Reciprocal propogation:

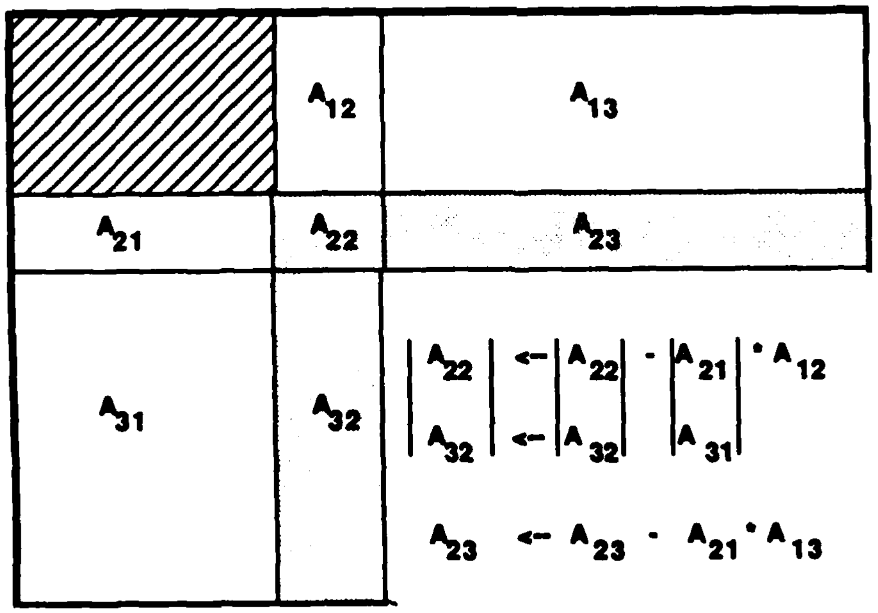
$$A_{32} \;\texttt{<---}\; a_{22} \, A_{32} \qquad (5)$$

B. M*M-based Algorithms

Another matrix partition permits the factorization to be performed on submatrices (Figure 2b). The equations equivalent to (2) - (5) are


A_{12}
A_{13}
a_{22}
A_{23}
A_{21}
A_{31}
A_{32}
$\begin{vmatrix} a_{22} \\ A_{32} \end{vmatrix} \leftarrow \begin{vmatrix} a_{22} \\ A_{32} \end{vmatrix} - \begin{vmatrix} A_{21} \\ A_{31} \end{vmatrix} * A_{12}$
$A_{23} \leftarrow A_{23} - A_{21} * A_{13}$

(a) M*V-based factorization kernel


A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
$\begin{vmatrix} A_{22} \\ A_{32} \end{vmatrix} \leftarrow \begin{vmatrix} A_{22} \\ A_{32} \end{vmatrix} - \begin{vmatrix} A_{21} \\ A_{31} \end{vmatrix} * A_{12}$
$A_{23} \leftarrow A_{23} - A_{21} * A_{13}$

(b) M*M-based factorization kernel

Figure 2. Factorization Kernels

Matrix-matrix multiply:

$$\begin{vmatrix} A_{22} \\ A_{32} \end{vmatrix} \leftarrow\!\!- \begin{vmatrix} A_{22} \\ A_{32} \end{vmatrix} - \begin{vmatrix} A_{21} \\ A_{31} \end{vmatrix} A_{12} \qquad (6)$$

$$A_{23} \leftarrow\!\!- A_{23} - A_{21} \; A_{13} \qquad (7)$$

Factorization:

$$A_{22} \leftarrow\!\!- L_{22} \; U_{22} \qquad (8)$$

Substitution:

$$A_{32} \leftarrow\!\!- A_{32} \; U_{22}^{-1} \; L_{22}^{-1} \qquad (9)$$

Alternatively, Eq. (9) can be replaced by two substitution steps

$$A_{32} \leftarrow\!\!- A_{32} \; U_{22}^{-1} \qquad (9a)$$

$$A_{23} \leftarrow\!\!- L_{22}^{-1} \; A_{23} \qquad (9b)$$

The advantage of Eq. (9) with a local memory will be discussed below.

The critical size parameter is the dimension of the square diagonal block (n_d). This has been chosen to be 64, the maximum vector length of the CRAY-2, and a length consistent with local memory size (see below).

C. Multiply Kernel Partitioning and Performance

1. Block partitioning

Because of the restricted size of LM, it is not feasible to load A_{32} and A_{23} into local memory. Therefore, submatrices are defined , viz

$$A_{ij} = \begin{vmatrix} A_{ij,11} & A_{ij,12} & \cdots & A_{ij,1r} \\ A_{ij,21} & A_{ij,22} & \cdots & A_{ij,2r} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ A_{ij,t1} & A_{ij,t2} & \cdots & A_{ij,tr} \end{vmatrix} \qquad (10)$$

The two dimensions of these component matrices - except at boundaries of **A** - is chosen to be n_d; with $n_d = 64$, up to three such matices can be stored in the 16K LM, with room left for system and temporary storage.

The components of Eqs. (6) and (7) are computed by a block dot product, viz

$$A_{23,1j} \leftarrow A_{23,1j} - \sum_{k=1}^{s} A_{21,1k} \; A_{13,kj} \qquad (11)$$

The complexity of such $n_d x n_d$ block multiplies is $O((n/n_d)^3)$ for the entire factorization algorithm.

2. Data transfer overhead

The total number of inter-memory data transfers to form Eq. (11) is $2(1+s)n_d^2$. With a clock period of tc, a transfer rate of W words/clock and a floating point execution rate of R_0 for data resident in LM, the multiply execution rate to produce $\mathbf{A_{23,ij}}$ above is

$$R = \frac{R_0}{1 + \frac{1+s}{s} \, \frac{tc\,R_0}{W\,n_d}} \qquad (12)$$

With $R_0 = 430$ MFLOPS, W = .8 (using CAL-coded transfer routines), and tc = 4.1 nsec, then $402 < R < 415$ MFLOPS for $1 < s < \infty$.

3. Conflict sensitivity

The presence of bank conflicts affects the transfer rate W. It is therefore informative to compute a normalized fractional sensitivity

$$S = (dR/R)/(dW/W)$$

$$= \frac{1}{1 + \frac{s}{1+s}\,\frac{W\,n_d}{tc\,R_0}} \qquad (13)$$

For the above values, $.064 > S > .033$ for $1 < s < \infty$. Thus, with $dW/W = -.5$ (an average delay of 50%, not necessarily representative), reductions in R of approximately 12 MFLOPS are predicted. The block multiply rate is now $390 < R < 403$ MFLOPS for $1 < s < \infty$. This execution rate is far greater than might be expected from a multiply code executing from CM with the same interference.

D. Block Factorization and Substitution

Both of the substitution steps of Eq. (9) can be performed with a sinlge matrix load of LM provided that $\mathbf{A_{32}}$ is partitioned as above, e.g.,

$$A_{32,i1} \;\leftarrow\!\!-\!\!- \; A_{32,i1}\, U_{22}^{-1}\, L_{22}^{-1} \qquad (14)$$

The execution rates of CAL-coded LU factorization and substitution steps of Eqs. (8) and (14) operating from local memory with $n_d = 64$ have been measured as 124 and 200 MFLOPS, respectively, including the effects of memory transfers and bank conflicts during a daytime load..

E. Overall Performance

Figure 3 gives the performance of a CAL-coded M*M-based factorization utilizing LM in comparison with Fortran- and CAL-coded M*V-based algorithms executing from CM. (Recall that LM M*V kernels are inefficient and so are not studied.) These were run during a daytime load at MFECC, using the CIVIC Fortran compiler circa November 14, 1985. The performance of a standard Gauss column-based Fortran code is also shown.

The Fortran Gauss algorithm (Appendix A) does not permit unrolling or other techniques for overcoming the hardware disadvantages of no chaining and a long CM path; the result is that no overlapping can be achieved between the three vector memory accesses, the vector multiply, or the vector add that characterize the inner loop. A rate of 60 MFLOPS for n=2048 is the result. This is likely the asymptotic rate of any Fortran factorization not utilizating loop unrolling or LM (see Appendix B).

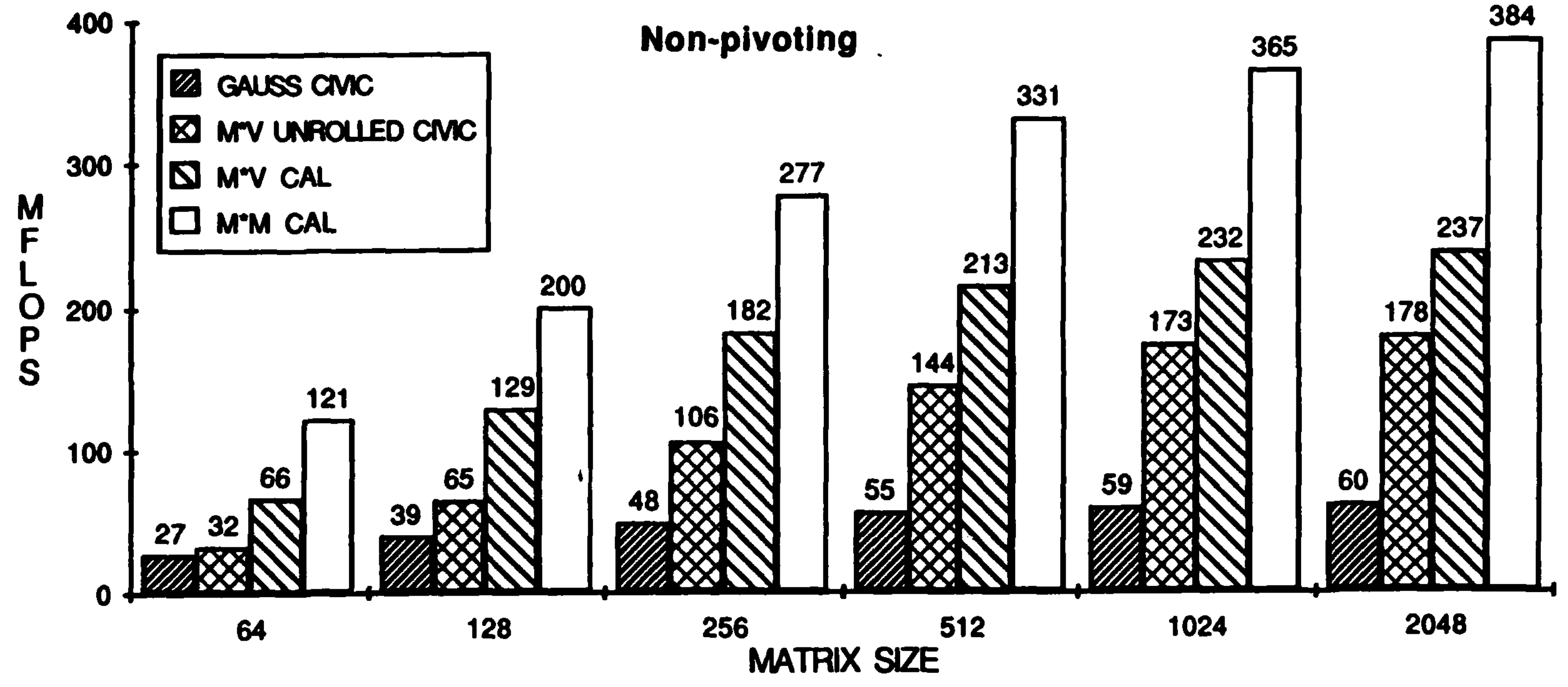


Figure 3. Performance of non-pivoting factorization

The Fortran M*V-based algorithm performances for n=2048 show a speedup of 2.96:1 over Gauss, principally due to a 16-way unrolling of the matrix-vector multiply. (The unrolling issue is studied in Appendix B in detail). Assembly coding yields another 1.62:1 speedup over Fortran, due in part to reduction of scalar operations associated with unrolling and to the resistance to bank conflicts which can be achieved in CAL by pre-fetching vector operands.

The LM M*M-based factorization ranges in performance between 124 MFLOPS for n=64 - when only the factorization of Eq. (8) must be performed - to nearly 400 MFLOPS when n=2048 and Eq. (11) is dominant. The 200-MFLOP performance of the substitution step of Eq. (8) maintains a relatively high performance vis-a-vis the other implementations for intermediate values of n.

III. PIVOTING ALGORITHMS AND PERFORMANCE

A. Influence of Pivoting

On a vector machine such as the CRAY-2, partial column pivoting has two components: (1) the search for the maximum element of a column, and (2) exchange of two complete rows of the matrix. The latter is usually preferred over maintenance of an index pointer in order to avoid relatively slow indirect addressing. These two functions are denoted

$$a \quad \longleftarrow \quad \mathrm{piv}\{\, s, V \,\}$$

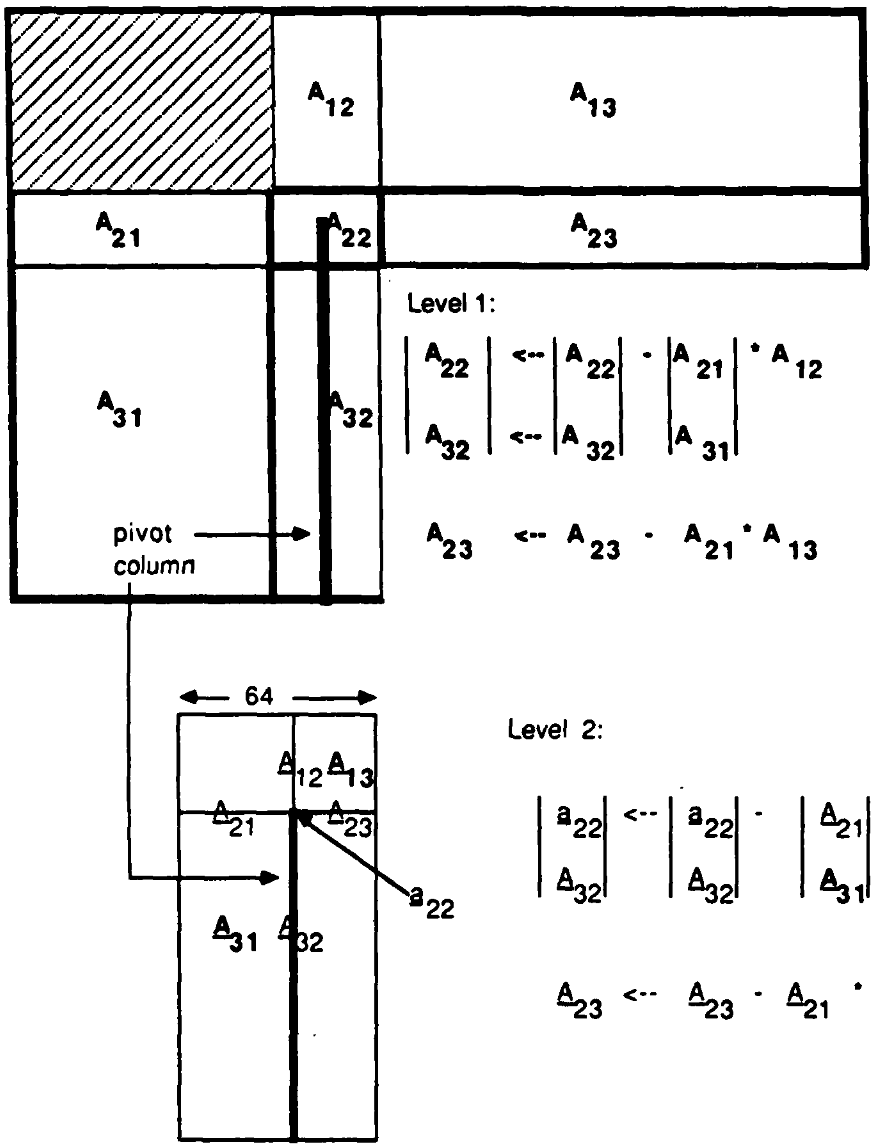
where *a* is the element of maximum absolute value of scalar s and the elements of vector V.

In M*V-based factorization this search is routinely performed after Eq. (2) or (3) by the step

$$a_{22} \quad \longleftarrow \quad \mathrm{piv}\{\, a_{22}, A_{32} \,\} \qquad (3a)$$

However, in the M*M-based version, the granularity of the algorithm does not recognize individual matrix elements and columns. The problem then becomes to <u>preserve the high performance of the matrix-matrix multiply by performing the majority of computations at the block-level, yet to occasionally expose individual columns to permit pivoting</u>. The solution is the following 2-level algorithm (Figure 14).

Equations (6) and (7) are carried out, viz



Level 1:

$$\begin{vmatrix} A_{22} \\ A_{32} \end{vmatrix} \leftarrow \begin{vmatrix} A_{22} \\ A_{32} \end{vmatrix} - \begin{vmatrix} A_{21} \\ A_{31} \end{vmatrix} * A_{12}$$

$$A_{23} \leftarrow A_{23} - A_{21} * A_{13}$$

Level 2:

$$\begin{vmatrix} a_{22} \\ \underline{A}_{32} \end{vmatrix} \leftarrow \begin{vmatrix} a_{22} \\ \underline{A}_{32} \end{vmatrix} - \begin{vmatrix} \underline{A}_{21} \\ \underline{A}_{31} \end{vmatrix} * \underline{A}_{12}$$

$$\underline{A}_{23} \leftarrow \underline{A}_{23} - \underline{A}_{21} * \underline{A}_{13}$$

Figure 4. 2-level pivoting algorithm.

Matrix-matrix multiply:

$$\left|\begin{matrix} \mathbf{A}_{22} \\ \mathbf{A}_{32} \end{matrix}\right| \longleftarrow \left|\begin{matrix} \mathbf{A}_{22} \\ \mathbf{A}_{32} \end{matrix}\right| - \left|\begin{matrix} \mathbf{A}_{21} \\ \mathbf{A}_{31} \end{matrix}\right| \mathbf{A}_{12} \qquad (15)$$

$$\mathbf{A}_{23} \longleftarrow \mathbf{A}_{23} - \mathbf{A}_{21} \; \mathbf{A}_{13} \qquad (16)$$

as an $O((n/n_d)^3)$ process. The columns of the resulting $\mathbf{A}_x = [\, \mathbf{A}_{22} \quad \mathbf{A}_{32} \,]^T$ block-column matrix are at this point partially reduced, with all the accumulations from the columns of $\mathbf{A}_{31}$ performed but without contributions from the internal columns of $\mathbf{A}_x$. $\mathbf{A}_x$ is then reduced using either a Gauss column reduction or the M*V method of Eqs. (2)-(5), viz (an underline represents components of this second reduction level)

$$\left|\begin{matrix} \underline{a}_{22} \\ \underline{A}_{32} \end{matrix}\right| \longleftarrow \left|\begin{matrix} \underline{a}_{22} \\ \underline{A}_{32} \end{matrix}\right| - \left|\begin{matrix} \underline{A}_{21} \\ \underline{\mathbf{A}}_{31} \end{matrix}\right| \underline{A}_{12} \qquad (17)$$

$$\underline{A}_{23} \longleftarrow \underline{A}_{23} - \underline{A}_{21} \; \underline{\mathbf{A}}_{13} \qquad (18)$$

$$\underline{a}_{22} \longleftarrow \mathrm{piv}\,\{\, \underline{a}_{22} \,,\, \underline{A}_{32} \,\} \qquad (19)$$

$$\underline{a}_{22} \longleftarrow 1/\underline{a}_{22} \qquad (20)$$

$$\underline{A}_{32} \longleftarrow \underline{a}_{22} \;\; \underline{A}_{32} \qquad (21)$$

The computations of Eq. (17-21) are performed from CM and will so be slowed by memory access delays.

Exclusive of the pivoting of Eq. (19), the result of these level-2 steps is the equivalent of factoring $\mathbf{A}_{22}$ in Eq. (8) and performing $\mathbf{A}_{32}\,(\mathbf{U}_{22})^{-1}$ in Eq. (9a). The substitution with $\mathbf{L}_{22}$ may then be performed either on the resultant $\mathbf{A}_{32}$ as in Eq. (9) or on $\mathbf{A}_{23}$ as in Eq. (9b). In either case, this can be carried out in local memory at a speed somewhat less than the 200 MFLOPS noted above.

With n_d fixed, the complexity of level 2 is readily shown to be $O(n^2)$, whereas the M*M kernel complexity remains $O((n/n_d)^3)$. For large n, the execution rate should therefore approach that of the multiply kernel or approximately 400 MFLOPS.

B. Implementation and Performance

Figure 5 presents the results of the same algorithms as Figure 3 but with pivoting. The rise of the M*M algorithm to the asymptotic rate is now slower. Several explanations are offered for this performance.

(a) M*M vs M*V CAL performance. Without pivoting, the advantage of M*M-based factorization was maintained for all n; in Figure 5 this occurs only for large n. This is explained in part by the complete CAL-coding of the diagonal block M*M-based factorization of Eq. (8) without pivoting (Figure 3), and the mixed Fortran- and CAL-coding of this step in both M*V-based factorization - with and without pivoting - and in level 2 of the M*M-based pivoting algorithm. With a mixed coding, all matrix-vector multiplies and searches for the maximum element of a column are performed in CAL, but the subroutine linkage to these routines introduces an overhead with O(n) complexity. Thus, comparisons in Figure 3 for small n include the effects of different codings, whereas those of Figure 5 do not.

(b) Effect of 2-level algorithm. The CRAY-2 implementation the piv{ s, V } function of Eq. (3a) requires a fixed overhead dependent only on the length of V and independent of the matrix element values. Consequently, it is possible to delineate between the pivoting speedown due to piv{ s, V } and that due to the 2-level nature of the algorithm. These are presented in Figure 6 for the M*M algorithm. For $n \geq 256$ - where the effect of the coding differences of (a) is largely dissipated - the larger degradation is the result of the piv { s, V } function. Since the latter cannot be avoided, the algorithmic speedown from the introduction of a second level does not appear significant.

IV. CONCLUSIONS

In general, the partitioning of an algorithm into larger computational tasks favors a parallel implementation since fewer task startups are involved. However, in a CRAY-2 system dedicated to an equation solution, equalizing the workload among the processors (load-leveling) also becomes an issue; this favors smaller tasks associated with M*V-based factorization. These issues will be investigated in a companion report.

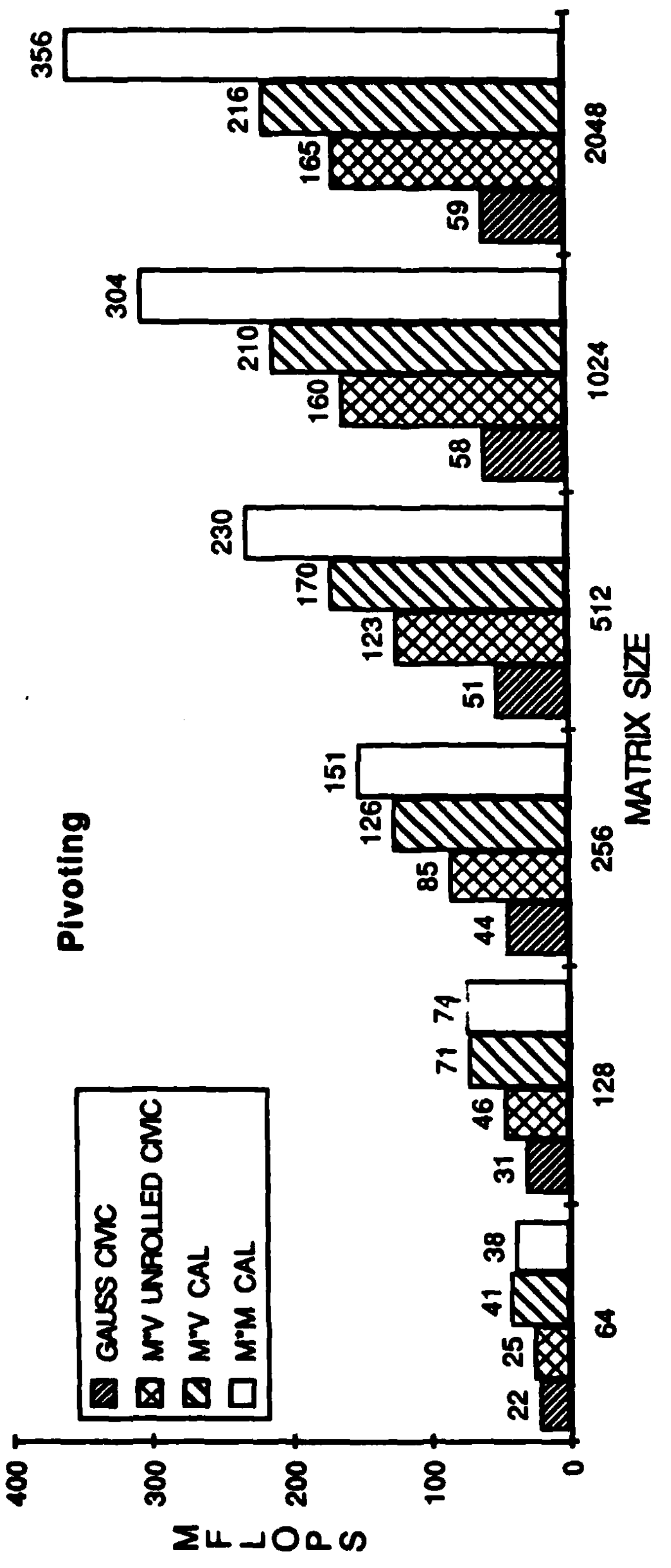


Figure 5. Performance of pivoting factorization

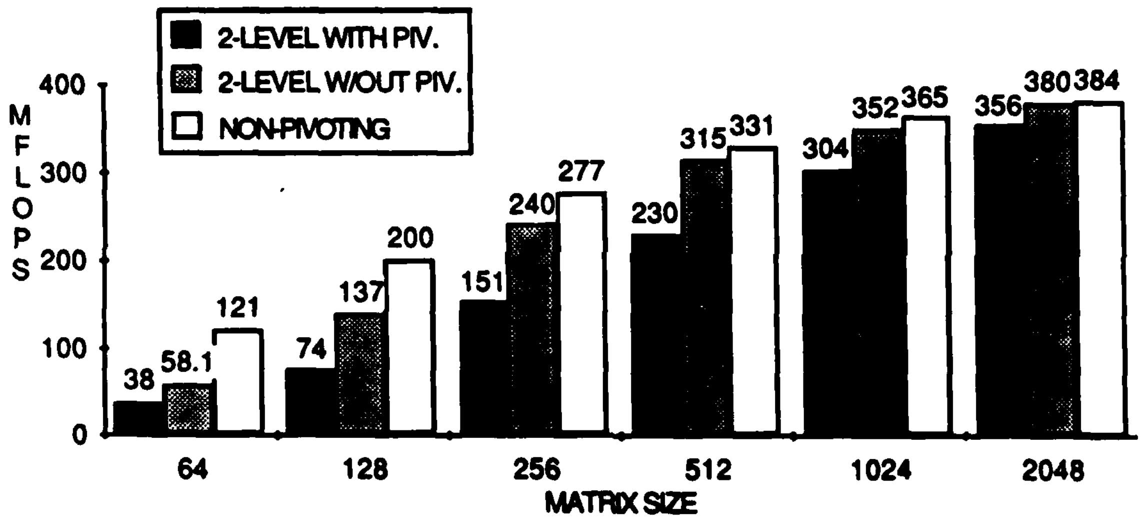


Figure 6. Effect of pivoting components

REFERENCES

[1] McKellar, A. C., and E. G. Coffman, "Organizing Matrices and Matrix Operations for Paged Memory Systems," CACM, vol. 12, no. 3, March, 1969, pp153-155.

[2] Von Fuchs, G., J. R. Roy, and E. Schrem, "Hypermatrix Solution of Large Sets of Symmetric Positive Definite Linear Equations," Comput. Math Appl. Mech. Engring., vol. 1, 1972, pp197-216.

[3] Calahan, D. A., "A Block-Oriented Sparse Equation Solver for the CRAY-1," Proc. 1979 Intl. Conf. on Parallel Processing, Bellaire, MI, pp234-239.

[4] Liu, P. S., and T. Y. Young, "VLSI Array Design Under Constraint of Limited I/O Bandwidth," Trans. IEEE, vol. C-32, no. 12, December, 1983, pp1160-1170.

[5] Calahan, D. A., "Preliminary Report on Results of Matrix Benchmarks on Vector Processors," Report #96, Systems Engineering Laboratory, University of Michigan, May, 1976.

[6] Fong, K. and T. Jordan, "Some Linear Algebraic Algorithms and Their Performance on the CRAY-1," Report LA-7664, Los Alamos Scientific Laboratory, June, 1977.

[7] Dongarra, J. J., and S. C. Eisenstat, "Squeezing the Most out of an Algorithm in CRAY Fortran," Report ANL/MCS-TM-9, Mathematics and Computer Science Division, Argonne National Laboratory, May, 1983; also in ACM Trans. on Mathematical Software, vol. 10, no. 3, pp221-230, 1984.

[8] Dongarra, J. J., F.G. Gustavson, and A. Karp, "Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine," SIAM Review, vol. 26, pp91-112, 1984.

[9] Dongarra, J. J., J. DU Croz, S. Hammarling, and R. J. Hanson, "A Proposal for an Extended Set of Fortran Basic Linear Algebra Subprograms," Report ANL/MCS-TM-41, Mathematics and Computer Science Division, Argonne National Laboratory, December, 1984.

[10] Saad, Youcef, "Communication Complexity of the Gaussian Elimination Algorithm on Multiprocessors," Report YALEU/DCS/RR-348, Department of Computer Science, Yale University, January, 1985.

[11] Calahan, D.A., P.L. Berry, G.C. Carpenter, K.B. Elliott, U.M. Fayyad and C.M. Hsiao. "MICHPAK: A Scientific Library for the CRAY-2," Report SARL #8, Supercomputer Algorithm Research Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, December 1, 1985.

APPENDIX A

FORTRAN PROGRAM LISTINGS

```
 C*****  GAUSS FACTORIZATION
        IMPLICIT REAL(A - H,O - Z)
>       DIMENSION TEMP(2048), A(2049,2048)
        NDIM = 2049
>       CALLLINK("UNIT6=TERMINAL//")
>       CALLLINK("UNIT5=TERMINAL//")
>C****  READ MATRIX SIZEDL
>  10 WRITE (5,20)
>  20 FORMAT (' ENTER MATRIX SIZE')
>       READ (5,30) N
>  30 FORMAT (I5)
>C****   FORMULATE DIAGONALLY-DOMINANT MATRIX
>       DO 50 I = 1, N
>         DO 40 J = 1, N
>  40     A(I,J) = -N + IABS(I - J)
>  50 A(I,I) = 1.1 * ((N - 1)*N - (I - 1)*I/2 - (N - I + 1)*(N - 1)/2)
>       T1 = SECOND(0)
>       CALL FAC(N, NDIM, A, TEMP)
>       T1 = SECOND(0) - T1
>       AN = N
>       AN2 = AN * AN
>       AN3 = AN2 * AN
>       OP = ((2.*AN3)/3.) - (AN2/2.) + (5.*AN/6.)
>       FLOPS = OP / T1
>       WRITE (5,60) A(N,N)
>       WRITE (5,60) T1, OP, FLOPS
>  60 FORMAT (3E14.6)
>       GO TO 10
>       END
>       SUBROUTINE FAC(N, NP1, A, TEMP)
        IMPLICIT REAL(A - H,O - Z)
        DIMENSION A(NP1,1), TEMP(1)
>       NM1 = N - 1
>       IF (NM1 .EQ. 0) GO TO 80
>       DO 70 J = 1, NM1
>  10     NMJP1 = N - J + 1
>         JJ = ISAMAX(NMJP1,A(J,J),1) + J - 1
>         T = ABS(A(JJ,J))
>         IF (T .EQ. 0) WRITE (6,20)
>  20     FORMAT (' ZERO PIVOT')
>         DO 30 I = 1, N
>  30     TEMP(I) = A(J,I)
>CDIR$ IVDEP
>         DO 40 I = 1, N
>  40     A(J,I) = A(JJ,I)
>         DO 50 I = 1, N
>  50     A(JJ,I) = TEMP(I)
>C***    RECIPROCATE DIAGONAL
>         A(J,J) = 1. / A(J,J)
>         ALPHA = A(J,J)
>         IJ = J + 1
>C***    PROPOGATE RECIPROCATED DIAGONAL DOWN COLUMN
>         DO 60 I = IJ, N
>  60     A(I,J) = ALPHA * A(I,J)
>C***    MULTIPLY COL. BY ELEMENT & SUBTRACT FROM ANOTHER COL.
>         DO 70 I = IJ, N
>           ALPHA = A(I,J)
>CDIR$ IVDEP
>           DO 70 K = IJ, N
>  70 A(I,K) = A(I,K) - ALPHA * A(J,K)
>C***    RECIPROCATE LAST DIAGONAL
>  80 A(N,N) = 1. / A(N,N)
>       RETURN
>       END
```

Table A-1. Gauss factorization

```
>C****  FACTORIZATION USING 16-WAY UNROLLED MATRIX-VECTOR MULTIPLIES
>      IMPLICIT REAL(A - H,O - Z)
>      DIMENSION TEMP(1024), A(1025,1024)
>      CALLLINK( "UNIT5 = TERMINAL // ")
>      CALLLINK( "UNIT6 = TERMINAL // ")
>      NDIM = 1025
>  10  WRITE (5,20)
>  20  FORMAT (' ENTER MATRIX SIZE')
>      READ (5,30) N
>  30  FORMAT (I5)
>C***    FORMULATE DIAGONALLY-DOMINANT MATRIX
>      DO 50 I = 1, N
>        DO 40 J = 1, N
>  40    A(I,J) = -N + IABS(I - J)
>  50  A(I,I) = 1.1 * ((N - 1)*N - (I - 1)*I/2 - (N - I + 1)*(N - I)/2)
>  60  T2 = SECOND(0)
>      DO 140 J = 1, N
>        IF (J .EQ. 1) GO TO 70
>        CALL SMXPY(A(1,J), A(J,J), A(J,1), N - J + 1, J - 1, NDIM)
>C**     SEARCH FOR PIVOT
>C2     CONTINUE
>C      FORM JTH ROW OF U
>  70    NMJP1 = N - J + 1
>        JJ = ISAMAX(NMJP1,A(J,J),1) + J - 1
>        T = ABS(A(JJ,J))
>        IF (T .EQ. 0) WRITE (6,80)
>  80    FORMAT (' ZERO PIVOT')
>        DO 90 I = 1, N
>  90    TEMP(I) = A(J,I)
>CDIR$ IVDEP
>        DO 100 I = 1, N
> 100    A(J,I) = A(JJ,I)
>        DO 110 I = 1, N
> 110    A(JJ,I) = TEMP(I)
>        A(J,J) = 1. / A(J,J)
>        IF (J .EQ. N) GO TO 150
>        IF (J .EQ. 1) GO TO 120
>        CALL SXMPY(A(J,1), A(J,J + 1), A(1,J + 1), N - J, J - 1, NDIM,
>     1       NDIM)
> 120    T = A(J,J)
>        JP1 = J + 1
>        DO 130 I = JP1, N
> 130    A(J,I) = T * A(J,I)
> 140  CONTINUE
> 150  T1 = SECOND(0) - T2
>      AN = N
>      AN2 = AN * AN
>      AN3 = AN2 * AN
>      OP = ((2.*AN3)/3.) - (AN2/2.) + (5.*AN/6.)
>      FLOPS = OP / T1
>      WRITE (5,160) T1, OP, FLOPS
> 160  FORMAT (3E13.4)
>      GO TO 10
>      END
```

Table A-2. Matrix-vector multiply-based factorization

```
C****  TRANSPOSED UNROLLED MATRIX-VECTOR MULTIPLY
>      SUBROUTINE SXMPY(X, Y, M, N1, N2, NDCM, NDXY)
>      REAL X(NDXY,1), Y(NDXY,1), M(NDCM,1)
>      J = MOD(N2,2)
>      IF (J .LT. 1) GO TO 20
>      DO 10 I = 1, N1
>   10 Y(1,I) = (Y(1,I)) - X(1,J) * M(J,I)
>   20 J = MOD(N2,4)
>      IF (J .LT. 2) GO TO 40
>      DO 30 I = 1, N1
>   30 Y(1,I) = ((Y(1,I)) - X(1,J - 1)*M(J - 1,I)) - X(1,J) * M(J,I)
>   40 J = MOD(N2,8)
>      IF (J .LT. 4) GO TO 60
>      DO 50 I = 1, N1
>   50 Y(1,I) = ((((Y(1,I)) - X(1,J - 3)*M(J - 3,I)) - X(1,J - 2)*M(J -
>     12,I)) - X(1,J - 1)*M(J - 1,I)) - X(1,J) * M(J,I)
>   60 J = MOD(N2,16)
>      IF (J .LT. 8) GO TO 80
>      DO 70 I = 1, N1
>   70 Y(1,I) = ((((((((Y(1,I)) - X(1,J - 7)*M(J - 7,I)) - X(1,J - 6)*M(
>     1J - 6,I)) - X(1,J - 5)*M(J - 5,I)) - X(1,J - 4)*M(J - 4,I)) - X(1,
>     2J - 3)*M(J - 3,I)) - X(1,J - 2)*M(J - 2,I)) - X(1,J - 1)*M(J - 1,
>     3I)) - X(1,J) * M(J,I)
>   80 JMIN = J + 16
>      IF (JMIN .GT. N2) GO TO 100
>      DO 100 J = JMIN, N2, 16
>        DO 90 I = 1, N1
>   90   Y(1,I) = ((((((((((((((((Y(1,I)) - X(1,J - 15)*M(J - 15,I)) - X(
>     1  1,J - 14)*M(J - 14,I)) - X(1,J - 13)*M(J - 13,I)) - X(1,J - 12)*
>     2  M(J - 12,I)) - X(1,J - 11)*M(J - 11,I)) - X(1,J - 10)*M(J - 10,
>     3  I)) - X(1,J - 9)*M(J - 9,I)) - X(1,J - 8)*M(J - 8,I)) - X(1,J -
>     4  7)*M(J - 7,I)) - X(1,J - 6)*M(J - 6,I)) - X(1,J - 5)*M(J - 5,I))
>     5  -X(1,J - 4)*M(J - 4,I)) - X(1,J - 3)*M(J - 3,I)) - X(1,J - 2)*M(
>     6  J - 2,I)) - X(1,J - 1)*M(J - 1,I)) - X(1,J) * M(J,I)
> 100 CONTINUE
>      RETURN
>      END
>C****  UNROLLED MATRIX-VECTOR MULTIPLY
>      SUBROUTINE SMXPY(X, Y, M, N1, N2, NDIM)
>      REAL X(1), Y(1), M(NDIM,1)
>      J = MOD(N2,2)
>      IF (J .LT. 1) GO TO 20
>      DO 10 I = 1, N1
>   10 Y(I) = (Y(I)) - X(J) * M(I,J)
>   20 J = MOD(N2,4)
>      IF (J .LT. 2) GO TO 40
>      DO 30 I = 1, N1
>   30 Y(I) = ((Y(I)) - X(J - 1)*M(I,J - 1)) - X(J) * M(I,J)
>   40 J = MOD(N2,8)
>      IF (J .LT. 4) GO TO 60
>      DO 50 I = 1, N1
>   50 Y(I) = ((((Y(I)) - X(J - 3)*M(I,J - 3)) - X(J - 2)*M(I,J - 2)) -
>     1X(J - 1)*M(I,J - 1)) - X(J) * M(I,J)
>   60 J = MOD(N2,16)
>      IF (J .LT. 8) GO TO 80
>      DO 70 I = 1, N1
```

Table A-2. Continued

```
   70 Y(I) = (((((((Y(I)) - X(J - 7)*M(I,J - 7)) - X(J - 6)*M(I,J - 6))
>    1 - X(J - 5)*M(I,J - 5)) - X(J - 4)*M(I,J - 4)) - X(J - 3)*M(I,J -
>    23)) - X(J - 2)*M(I,J - 2)) - X(J - 1)*M(I,J - 1)) - X(J) * M(I,J)
>  80 JMIN = J + 16
>       IF (JMIN .GT. N2) GO TO 100
>       DO 100 J = JMIN, N2, 16
>         DO 90 I = 1, N1
>  90     Y(I) = ((((((((((((((((Y(I)) - X(J - 15)*M(I,J - 15)) - X(J -
>    1  14)*M(I,J - 14)) - X(J - 13)*M(I,J - 13)) - X(J - 12)*M(I,J -
>    2  12)) - X(J - 11)*M(I,J - 11)) - X(J - 10)*M(I,J - 10)) - X(J -
>    3  9)*M(I,J - 9)) - X(J - 8)*M(I,J - 8)) - X(J - 7)*M(I,J - 7)) -
>    4  X(J - 6)*M(I,J - 6)) - X(J - 5)*M(I,J - 5)) - X(J - 4)*M(I,J -
>    5  4)) - X(J - 3)*M(I,J - 3)) - X(J - 2)*M(I,J - 2)) - X(J - 1)*M(
>    6  I,J - 1)) - X(J) * M(I,J)
> 100 CONTINUE
>       RETURN
>       END
```

Table A-2. Continued

```
>C****  CIVIC2 BLOCKED FACTORIZATION & SOLUTION WITH PIVOTING
>C*       THIS VERSION HAS ASSEMBLY-CODED SLM3D.LLM3D
>C*       MXMPMA,MXVPVA,SUBPIV, AND ISAMAX ROUTINES
>      IMPLICIT REAL(A - H,O - Z)
>      INTEGER QDIM
>      DIMENSION B(2048), AS(1,1), TEMP(2048), A(2049,2048)
>      DIMENSION TIME(4)
>      REGFILELM1, LM2, LM3
>      COMMON /LM1/ AT(64,64)
>      COMMON /LM2/ Q(64,64)
>      COMMON /LM3/ SUB(64,64)
>      CALLLI NK("UNIT6=TEMRINAL // ")
>      CALLLI NK("UNIT5=TERMINAL // ")
>      NDIM = 2049
>      QDIM = 64
>C****  READ MATRIX SIZE.BLOCK SIZE
>  10 WRITE (5,20)
>  20 FORMAT (' ENTER N, NSIZE.')
>      READ (5,30) N, NSIZE
>  30 FORMAT (2I5)
>C****  FORMULATE DIAGONALLY-DOMINANT MATRIX
>C*       AND RHS WITH SOLUTION B(J)=J+1
>      DO 50 I = 1, N
>        B(I) = 0.
>        DO 40 J = 1, N
>          A(I,J) = -N + IABS(I - J)
>  40    IF (I .NE. J) B(I) = B(I) + A(I,J) * (J + 1)
>        A(I,I) = 1.1 * ((N - 1)*N - (I - 1)*I/2 - (N - I + 1)*(N - I)/2)
>  50 B(I) = B(I) + A(I,I) * (I + 1)
>  60 NM1 = N - 1
>      DO 70 LL = 1, 4
>  70 TIME(LL) = 0.
>C****   MAIN FACTORIZATION ALGORITHM
>      DO 120 LL = 1, N, NSIZE
>        TIM1 = SECOND(0)
>        LLNS = MINO(LL + NSIZE - 1,N)
>        LLM1 = LL - 1
>        NDIAG = LLNS - LL + 1
>        CALL ALUPIV(A, B, TEMP, LL, LLNS, NDIM, N)
>        TIM2 = SECOND(0)
>        TIM3 = TIM2
>        TIM4 = TIM3
>C****  FORM (A23*L22**-1)
>  80    NMLLNS = N - LLNS
>        IF (NMLLNS .EQ. 0) GO TO 110
>C****  LOAD LM WITH L & U AS TWO SEPARATE ARRAYS REQUIRED BY SUBPIV
>        CALL LLM3D(A(LL,LL), 1, NDIAG, NDIAG, 1, 1, NDIM, 1, QDIM, 1)
>        CALL LLM3D(A(LL,LL), 2, NDIAG, NDIAG, 1, NDIM, 1, 1, QDIM, 1)
>        JX = LLNS + 1
>        LK = LLNS - LL
>        LKP1 = LK + 1
>        DO 100 J = JX, N, NSIZE
>          NMJ = MINO(N - J + 1,NSIZE)
>C****  LOAD BLOCK OF A23 FROM LM INTO LM
```

Table A-3. Blocked factorization with pivoting

```
>          CALL LLM3D(A(J,LL), 3, NMJ, LKP1, 1, 1, NDIM, 1, QDIM, 1)
>C****    SUBSTITUTE INTO BLOCK
>          CALL SUBPIV(NMJ, LK, QDIM)
>C****    STORE BLOCK FROM LM INTO CM
>  90      CALL SLM3D(A(J,LL), 3, NMJ, LKP1, 1, 1, NDIM, 1, QDIM, 1)
> 100    CONTINUE
>        TIM3 = SECOND(0)
>C****   INNER LOOP BLOCK MULTIPLIES
>        NMJX = N - JX + 1
>        LLNS1 = MIN0(LLNS + NSIZE,N)
>        CALL MXMPMA(A(JX,1), 1, NDIM, A(1,JX), 1, NDIM, A(JX,JX), 1,
>     1       NDIM, NMJX, LLNS, LLNS1 - LLNS, -1)
>        LLNS2 = N - LLNS1
>        IF (LLNS2 .EQ. 0) GO TO 110
>        CALL MXMPMA(A(JX,1), 1, NDIM, A(1,LLNS1 + 1), 1, NDIM,
>     1       A(JX,LLNS1 + 1), 1, NDIM, LLNS1 - LLNS, LLNS, LLNS2, -1)
> 110    TIM4 = SECOND(0)
>        TIME(1) = TIME(1) - TIM1 + TIM2
>        TIME(2) = TIME(2) - TIM2 + TIM3
>        TIME(3) = TIME(3) - TIM3 + TIM4
> 120 CONTINUE
>     TIM5 = SECOND(0)
>C**** FACTORIZATION ENDED:  FORWARD SUBSTITUTION
>     DO 130 LL = 1, N, NSIZE
>        LSAVE = LL
>        IF (LL .EQ. 1) GO TO 130
>        LLNS = MIN0(LL + NSIZE - 1,N) - LL + 1
>        CALL MXVPVA(A(LL,1), 1, NDIM, B, 1, B(LL), 1, LLNS, LL - 1, -1)
> 130 CONTINUE
>C****   BACK SUBSTITUTION HAS TWO STEPS
>C*       1ST STEP: M*B
>     LZ = LSAVE
>     DO 210 LM = 1, N, NSIZE
> 140    IF (LM .EQ. 1) GO TO 150
>        LL = LZ + NSIZE
>        NQ = N - LL + 1
>        CALL MXVPVA(A(LZ,LL), 1, NDIM, B(LL), 1, B(LZ), 1, NSIZE, NQ,
>     1       -1)
>C*       2ND STEP:  (U**-1 L**-1 B)
> 150    LLNS = MIN0(N,LZ + NSIZE - 1)
>        LLNSMZ = LLNS - LZ
>        IF (LLNSMZ .LE. 0) GO TO 170
>        LLNSM1 = LLNS - 1
>        DO 160 J = LZ, LLNSM1
>          JP1 = J + 1
>          DO 160 K = JP1, LLNS
> 160    B(K) = B(K) - A(K,J) * B(J)
> 170    B(LLNS) = B(LLNS) * A(LLNS,LLNS)
>        IF (LLNSMZ .LE. 0) GO TO 200
>        DO 190 J = LZ, LLNSM1
>          JJ = LZ + LLNS - J
>          JJM1 = JJ - 1
>          DO 180 K = LZ, JJM1
> 180    B(K) = B(K) - A(K,JJ) * B(JJ)
> 190    B(JJM1) = B(JJM1) * A(JJM1,JJM1)
> 200    LZ = LZ - NSIZE
```

Table A-3. Continued

```
> 210 CONTINUE
>     TIME(4) = SECOND(0) - TIM5
>C****  COMPUTE PERFORMANCE
>     CALL RESULT(TIME, N, NSIZE)
>C****   CHECK SOLUTION
>     DO 220 JK = 1, N
>       BJK = JK + 1
>       IF (ABS(B(JK) - BJK) .GT. 1.D-6) GO TO 230
> 220 CONTINUE
>     WRITE (5,250)
>     GO TO 260
> 230 WRITE (6,240) N, NSIZE, (B(JK),JK=1,N)
> 240 FORMAT (2I4, 4(1PE16.8)/(4(1PE16.8)))
>     STOP 2
> 250 FORMAT ('   OK')
> 260 CONTINUE
>     GO TO 10
>     END
>     SUBROUTINE ALUPIV(A, B, TEMP, LL, LLNS, NDIM, N)
>     DIMENSION TEMP(1), A(NDIM,1), B(1)
>     DO 90 K = LL, LLNS
>       KP1 = K + 1
>       JK = LLNS - K
>       KMLL = K - LL
>       NMK = N - K
>       NMKP1 = NMK + 1
>       IF (KMLL .EQ. 0) GO TO 10
>       CALL MXVPVA(A(K,LL), 1, NDIM, A(LL,K), 1, A(K,K), 1, NMKP1,
>    1          KMLL, -1)
>  10   KK = ISAMAX(NMKP1,A(K,K),1) + K - 1
>       T = ABS(A(KK,K ))
>       IF (T .EQ. 0) WRITE (6,20)
>  20   FORMAT (' ZERO PIVOT')
>       DO 30 I = 1, N
>  30   TEMP(I) = A(K,I)
>CDIR$ IVDEP
>       DO 40 I = 1, N
>  40   A(K,I) = A(KK,I)
>       DO 50 I = 1, N
>  50   A(KK,I) = TEMP(I)
>       TEM = B(K)
>       B(K) = B(KK)
>       B(KK) = TEM
>       A(K,K) = 1. / A(K,K)
>       IF (KMLL .EQ. 0 .OR. JK .EQ. 0) GO TO 70
>  60   CALL MXVPVA(A(LL,KP1), NDIM, 1, A(K,LL), NDIM, A(K,KP1), NDIM,
>    1          JK, KMLL, -1)
>  70   IF (NMK .EQ. 0) GO TO 90
>       DO 80 ILI = 1, NMK
>  80   A(K + ILI,K) = T * A(K + ILI,K)
>  90 CONTINUE
>     RETURN
>     END
>C****  THIS ROUTINE COMPUTES PERFORMANCE OF COMPONENTS OF
>C*         FACTORIZATION AND OF SOLUTION
>     SUBROUTINE RESULT(TIME, N, NSIZE)
```

Table A-3. Continued

```
>     DIMENSION TIME(1), FLOP(4), OP(4)
>     DO 10 J = 1, 4
>  10 OP(J) = 0.
>     DO 20 J = 1, N, NSIZE
>       AN = MIN(N - J + 1,NSIZE)
>       AN2 = AN * AN
>       AN3 = AN * AN2
>       OP(1) = OP(1) + ((2.*AN3)/3.) - (AN2/2.) + (5.*AN/6.) + RES *
>    1  AN2
A
>       RES = MAX0(0,N - J + 1 - NSIZE)
>       OP(2) = OP(2) + RES * (AN2 - AN)
>  20 OP(3) = OP(3) + 2. * RES * AN * RES
>     AN = N
>     OP(4) = (2.*AN*AN) - AN
>     DO 30 J = 1, 4
>       FLOP(J) = 0.
>  30 IF (TIME(J) .NE. 0.) FLOP(J) = OP(J) / TIME(J)
>     WRITE (5,40) (TIME(J),OP(J),FLOP(J),J=1,4)
>  40 FORMAT (3(1PE12.4))
>C****  TOTAL FACTORIZATION PERFORMANCE
>     TIME(1) = TIME(1) + TIME(2) + TIME(3)
>     OP(1) = OP(1) + OP(2) + OP(3)
>     FLOP(1) = OP(1) / TIME(1)
>     WRITE (5,40) TIME(1), OP(1), FLOP(1)
>     RETURN
>     END
```

Table A-3. Continued

APPENDIX B

UNROLLED MATRIX - VECTOR MULTIPLY PERFORMANCE

COMMENTS ON CHARACTERISTICS

The performance of Table B-1 was obtained between 5-8am on 12/10/85 at MFECC. The execution rates were obtained from averaging rates of 100-1200 runs of each code. As much as a 20% variation in average rates was noted in the 64-way unrolled code by running at different times of the day; in general, the average rates of Figure B-1 tend to be lower than rates measured at other times. Small differences in performance for different unrollings probably can be attributed to memory loading variations.

It should be noted that 2-way unrolling shows no advantage over 1-way unrolling for small matrices; 4-way unrolling shows a marked advantage for all sizes. The consistent degradation of 64-way vis-a-vis 32-way unrolling is not explained.

It was decided that 16-way unrolling offered a reasonable performance-complexity compromise, and was adopted for use in the Fortran M*V factorization codes of Figures 3 and 5. See Appendix A for listing of an unrolled multiply.

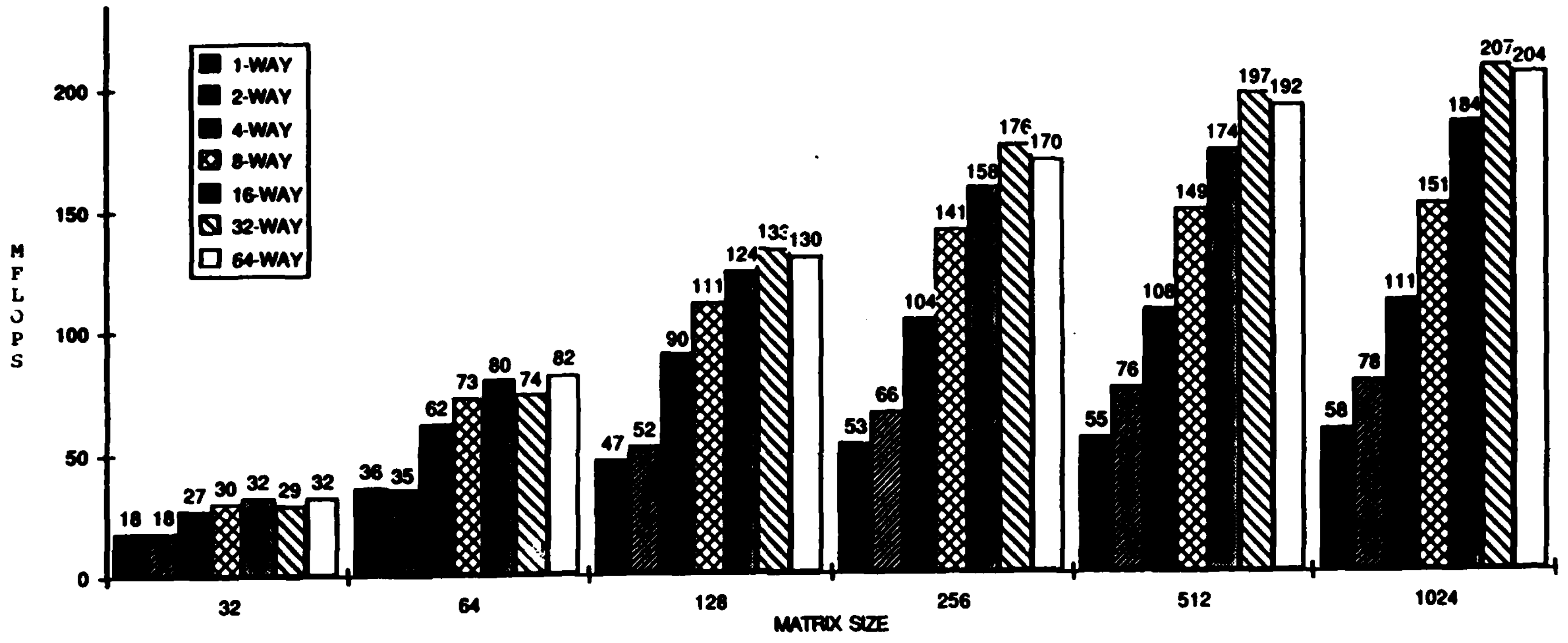


Figure B-1. Effect of unrolling on M*V performance